# Lab Overview

In this lab assignment, you will do the following:

- Add decode logic, an NVRAM (an EPROM substitute), and a status LED to the hardware developed in Lab #1. In this lab the NVRAM will be used for non-volatile code storage. Note that the same NVRAM will be used in Lab #3 as a standard SRAM for data storage.
- Write simple assembly programs to test NVRAM accesses and perform user I/O.
- Learn how to use timers and write ISRs in assembly.
- Learn how to use a device programmer for code storage.
- Learn how to use a logic analyzer to capture state and timing information.
- Add the AT89C51RC2 processor to your board and enable in-circuit programming via FLIP.
- Continue learning about the ARM architecture and the ARM dev board.

**Students must work individually and develop their own original and unique hardware/software.**

The signature due dates for this lab assignment are **Friday, September 29, 2023 (Part 1 Elements) and Friday, October 6, 2023 (Part 2 Elements)**. Note: Use the C501 processor for Part 1 Elements and the Atmel AT89C51RC2 processor for Part 2 Elements.

The submission due date for this lab is **11:59pm Sunday, October 8, 2023**.

The cutoff date for this lab is **Saturday, October 14, 2023**.

Labs completed after the signature due date or submitted after the submission due date will be accepted, but will receive grade reductions. Labs will not be accepted after the cutoff date.

This lab is weighted as 10% of your course grade.

Required elements are necessary in order to proceed to the next lab assignment. Supplemental elements of this lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the minimum requirements for the lab.

All items on the signoff sheet must be completed to get a signature, but partial credit is given for incomplete labs. Note that receiving a signature on the signoff sheet does not mean that your work is eligible for any particular grade; it merely indicates that you have completed the work at an acceptable level. Students should always submit any work completed no later than the cutoff date for the lab in order to receive some partial credit.

Note: Logic analyzers will be used for the rest of the semester starting with this lab assignment. There are a number of 34-channel Intronix LA1034 LogicPort logic analyzers available for students to sign out for the semester. Students can download the LogicPort software and help files from http://www.pctestinstruments.com. Students will need to take financial responsibility for any equipment they sign out, and must return the items in good semester by the final project submission date. The LogicPort analyzers are PC-based with a USB connection and are quite portable.

Note: All of the NVRAM chips included in the tool kits were individually checked in the device programmers by the TA's to verify that the chips are in good working condition. Take good care of these chips, since they're expensive (~$20-$25 each), and if you damage the chip you will need to pay for a replacement.

# Lab Details

1. Review Homework assignments #3, #4, #5, #6, #7.
2. Refer to Lab #1 regarding layout considerations, labeling, etc. All signals on all ICs must be labeled.
3. Solder in the 28-pin wire wrap sockets for both the EPROM and SRAM. Do **not** solder in the ZIF socket that is included in the tool kit - that ZIF socket needs to be returned at the end of the semester.
4. Design and implement your decoding circuitry so that your memory map looks like the following: Your NVRAM (EPROM) must be located starting at address 0000h and must occupy 32KB of address space (addresses 0000h-7FFFh). [Note that in future lab assignments, you will be adjusting your memory map.] The last 32KB of address space (addresses 8000h-FFFFh) should be reserved for use later in the semester. Options for your decode logic include the Atmel ATF16V8C SPLD (the SPLD is the preferred solution), discrete logic, a 74LS138, or a 74LS156. Use a 74LS373 to demultiplex the 8051 address/data bus.
5. Design and implement your NVRAM (EPROM) circuit. Your NVRAM (EPROM) must drive the data bus only during a microcontroller program read cycle. For this lab, the NVRAM should simulate an EPROM: it must not drive the bus during a microcontroller data read cycle, and it must not accept data during a microcontroller data write cycle. **If using the Atmel SPLD for decode logic, make sure you have an easy way to remove the SPLD chip from your system. You will likely need to reprogram it.**
6. Obtain a copy of the document which compares the Intel hex record format and the Motorola S-record format, and make sure you understand how hex records are used.
7. Read the documents available on the course web site regarding the device programmers we use in our laboratory. You will need to use these programmers with your SPLD, NVRAM, and processor.
8. Learn how to generate Motorola S-records and Intel hex records with the software tools in the lab, and how to program your NVRAM using one of the device programmers in the lab. Choose the correct NVRAM type (for the TI BQ4011YMA NVRAM, choose **Benchmarq** as the manufacturer and **BQ4011Y** as the device). Be able to verify that a device is blank before programming. Be able to verify that the contents of the NVRAM match the contents of the buffer on the PC after the NVRAM is programmed. **Be careful to insert your device into the device programmer correctly. Incorrect insertion will damage the (expensive) NVRAM.** Do not solder near the device programmer, as solder can easily damage the programmer electronics. When using the external device programmer (parallel port or USB), use only the power adapter specifically made for that particular programmer. Use of the wrong power adapter could damage the programmer.
9. Carefully push the 28-pin ZIF socket into your board's "EPROM" wire wrap socket (either orientation of the ZIF socket is fine - choose an orientation that eliminates any interference between the ZIF lever and components on your board). Again, do **not** solder in the ZIF socket that is included in the tool kit - that ZIF socket needs to be returned (in good condition) at the end of the semester.
10. [**Optional**] For initial hardware bring up, write a simple 'NOP CPL AJMP' infinite loop in assembly as shown on the hex record handout. Start at address 0000h and then jump to the loop, which loops at address 0021h. Verify that the microcontroller correctly executes this code out of the NVRAM. This will allow you to verify that fetches from the NVRAM are happening correctly and that the 8051 is correctly executing instructions. Your code should toggle an unconnected 8051 port pin to help you verify that your code is running properly. A probe can be used to check the pin output.
11. Design and implement a circuit which will allow you to drive an LED using one of the 8051 Port 1 or Port 3 pins. You may want to use a transistor and current limiting resistor in your design. Good choices for port pins are P1.1–P1.7.

**Note: Students in this course need to use a version control system for source code and other course design files. Options include Git, Subversion (SVN), or others. Students need to make their code for this course private and not share it with others.**

**Note: Use the C501 processor for Part 1 Elements and the Atmel AT89C51RC2 processor for Part 2 Elements.**

12. [**Part 1 Required Element**[1]] Write an assembly program which contains two parts; a main loop and an interrupt service routine (ISR). The main assembly code should first initialize the 8051 registers and then enter an infinite loop. An ISR triggered by Timer 0 must blink an LED (by toggling a port pin) at about 1.38 Hz +/- 0.4% (on for ~0.362 seconds and off for ~0.362 seconds). A second unused port pin must be toggled each time the ISR executes (set the port pin to a logic high as the first instruction in your ISR, and clear the port pin to a logic low immediately before you execute the RETI instruction).

    **[Recommended]** Choose a third unused port pin to use as an input and regularly/periodically retrieve the state of the pin in your assembly code. As long as the pin is at a HIGH state, keep the LED blinking at the rate shown above. However, whenever the pin is detected at a LOW state, the LED should blink at a slower rate of ~0.30 Hz. Hint: In Lab 2 Part 2 Required Element 18, you will use a header/jumper for the /EA input; you can use a similar method or just a wire that you temporarily connect to power or ground to provide a HIGH or LOW input to the input port pin in this element.

You can potentially debug some of your code using a simulator like Emily52 or EdSim51 so that you reduce the time you spend programming NVRAMs, but note that full timer and interrupt support is not present in our version of Emily52. The 'V' command allows you to vector to an ISR in Emily52. **You can also debug your code in an EFM8BB1 dev board.** [Note: After you get your RS-232 interface and flash-based processor working in this lab assignment, you will be able to program your processor while it is in the system.]

- **Using the instruction set summary tables (available in the programmer's guide or instruction set documents), calculate how long the ISR takes to execute once, assuming a clock frequency of 11.0592 MHz. You will likely have some conditional jumps in your ISR code, so make sure to calculate both the longest and shortest time it takes the ISR to execute.**
- **Compare the calculated ISR time to the time measured with the second port pin, which toggles at the beginning and at the end of each ISR execution. Do the calculated and measured times match? State your assumptions and explain any differences you see.**

13. [**Part 1 Required Element**[1]] Hook up the logic analyzer to the address bus (all 16 signals A[15:0]), data bus (all 8 signals D[7:0]), the control lines on the 8051, and the chip selects from the decode logic and capture fetches of instructions from the NVRAM. Be able to decode the data shown on the logic analyzer and prove that the fetched instructions match the contents of the NVRAM. Learn how to use both the state and timing modes of the logic analyzer (you may be quizzed on this, so practice this until you're good at using the logic analyzer).

- Using the state mode, capture a sequence of instructions and compare the sequence to the listing file for the code being executed. For the state clock, you can investigate using $\overline{PSEN}$, $\overline{READ}$, or ALE. **Before your demo to the TA, prepare one screen capture of the logic analyzer triggered on a fetch in state mode.**
- Using the timing mode, measure the time which elapses **from when the 74LS373 latches the address supplied by the 8051 to when the $\overline{PSEN}$ signal is** activated during an instruction fetch. **Before your demo to the TA, annotate a screen capture to show the measurement of $t_{LLPL}$ in timing mode and prove that your measured time meets the processor data sheet specification for $t_{LLPL}$.**
- **Show and discuss both of these screen captures with the TA during the signoff.**

14. **[Part 2 Required Element]:** Determine how to program the Atmel AT89C51RC2 using a device programmer (e.g. the Phyton ChipProg-48) to set the lowest security levels, to enable reset at address 0000h, and to enable the XRAM. Programming guide notes are available on the course web site. The first time you program the Atmel processor, **erase it first**; otherwise, you may see the device programmer report some errors.

    Replace the C501 on your board with the Atmel AT89C51RC2. Verify that the oscillator circuit and run-time reset circuit work correctly, like you did in Lab #1. The ALE signal should come up at the correct frequency after every power cycle and reset. Note: The AO bit of the AUXR register must be '0b' (its default value) in order for the processor to emit ALE like the C501 processor.

    Verify that your Lab #2 blinking LED code that is located in the NVSRAM still functions correctly with this new processor. /EA will still be low for this test.

    **NOTE:** As discussed in lecture, it is possible to use a supervisory circuit (e.g. Microchip MCP-101, Maxim MAX1232, etc.) to protect the processor Flash memory against corruption due to power supply brownout or shutdown issues. See Atmel application note "External Brown-out Protection for C51 Microcontrollers with Active High Reset Input". Note that the parts kit does not include a supervisory circuit; most students do not do this optional element.

15. **[Part 2 Required Element]:** Design and implement your RS-232 circuit utilizing the MAX232 driver/receiver. This circuitry is not memory mapped, but instead will use the RX and TX lines on Port 3 of the 8051. On your RS-232 connector, you may connect RTS to CTS, and you may connect DTR to DSR and DCD. Make sure to wire the ground pin on your connector to ground on your board.

    **TIP:** You can determine which pin on the cable is the TX pin from the PC by using a terminal emulator program and pressing and holding a key on the keyboard while probing the cable pins with an oscilloscope to see which pin is toggling.

    **NOTE:** Make sure you understand the DTE and DCE perspective on the serial communications link.

    **NOTE:** DO NOT probe the +10V and -10V RS-232 voltages with a digital logic probe.

    **NOTE:** If your personal computer does not have a serial port, you may use a USB to RS-232 converter (as provided in the course tools kit).

    Your parts kit should contain four 1.0uF caps suitable for the charge pump capacitors (C1, C2, C3, C4) for the MAX232 chip. You can use larger capacitors up to 10uF instead, if you desire. While some newer versions of the MAX232 chip can use smaller capacitors, some older versions of the chip prefer the larger caps. Using a smaller capacitor value saves space on your board; however, using a large capacitance value reduces ripple on the +10V and –10V charge pump outputs.

16. [**Optional**] This optional program just aids in making sure your system is wired and programmed correctly. Write an assembly program which initializes the 8051 serial port and then continuously (in an infinite loop) transmits the character 'U'. Using an oscilloscope, verify that the transmitted patterns correspond with the ASCII value for this character and that the baud rate is correct. Verify that the characters appear on screen.

    Make sure you understand the relation between bit rate and baud rate as well as between baud rate and the underlying carrier frequency.

    Now, modify the program to make it echo the characters it receives from the 8051 serial port. Every time a valid character is received, that same character should be transmitted back out the serial port.

    Learn how to configure a terminal emulator program (e.g. TeraTerm, RealTerm, PuTTY,…) on a host computer to allow you to communicate over the serial port. Be sure that you have the serial cable hooked up to the correct serial port on the computer, and that the terminal emulator is configured to use that same serial port. When first testing your hardware, you should configure the terminal emulator to communicate directly to the appropriate COM port on the PC at 9600 baud, 8 data bits, 1 stop bit, no parity and no flow control. **After you verify the hardware is working fine, you should increase the baud rate to the maximum rate that is reliable**.

17. **[Part 2 Required Element]:** Implement the required circuit features for /EA and /PSEN to enable the Atmel UART bootloader to execute when coming out of reset. Use a **momentary pushbutton and pull-down resistor** for /PSEN, and hold that button when coming out of reset in order to force bootloader operation; then, after the bootloader has started, release that button to eliminate drive fight issues on the /PSEN line. Use a header/jumper for the /EA input. Note that if you use these hardware conditions to enter the bootloader when you come out of reset, then the Atmel bootloader is entered regardless of the values of BLJB, BSB, and SBV.

    Verify that your Lab #2 blinking LED code runs correctly from the flash memory (/EA high) on the new processor. Verify that FLIP can communicate with the Microchip/Atmel bootloader. You can use FLIP to download the code to internal flash memory on the processor. Note that you may need to use a command line option (e.g. –I) with the assembler to create the Intel hex file needed by FLIP.

**NOTE:** Simulation of C code is completely optional, and most students in this course do not simulate their code once they start writing in C instead of assembly. If you want to simulate your assembly code using Emily52, note that SFR's are **not** emulated in our version of Emily52, so you can't simulate all features of the 8051, such as the real-time aspects of serial port operation, timers, and interrupts. However, you can still use the simulator to verify much of your code. Simulate interrupts in Emily52 by pressing 'v' for 'vector'. Alternatively, you can consider simulating your code on the EFM8 dev board.

18. **[Part 2 Required Element]:** Continue learning about your dev boards. Many reference documents and files are available on Canvas.

    **NOTE: The STM32 is still actively produced and supported by ST Microelectronics and therefore is a better long term tool than the MSP432. The MSP432 is a good educational tool, but would not be a solution for a future embedded systems product.**

    a) Learn how to create and build a simple project in the development environment (e.g. STM32CubeIDE, etc.) and download your executable to the ARM dev board. Utilize the tutorials & training that are available via the development environment and YouTube. **Note that students are highly encouraged to do bare metal programming in this course and not to rely on development tools that abstract the lower levels of the hardware and firmware. In this course, your work will be evaluated as having more engineering content if you are developing all the code rather than relying on a tool to develop the code for you.**

    b) Write a program to toggle the on-board red LED using a timer interrupt on the ARM processor.
    - Use SMCLK as clock source
    - Use Continuous mode or Up mode as the Timer mode
    - The on-time and off-time of the LED should be nearly 270 ms each. Try to verify the delay by first toggling a GPIO pin and verifying the delay on the oscilloscope and then replacing GPIO pin toggling with LED toggling in your code.
    - Note: The MSP432 reference code for this program is: msp432p401x_ta0_01

    c) Write a program to control on-board LEDs using a push-button on the ARM dev board.
    - Configure a timer to toggle the RGB LED between green and blue at a reasonable, visible rate. Each time the timer ISR fires, turn off the currently-lit LED and turn on the other LED. When your program initializes the hardware, the green LED should be lit first.
    - Use a GPIO interrupt to monitor push-button operation. Each time the button is pressed, have the GPIO ISR alternate between disabling and enabling the toggling of the LEDs.
    - Note: The MSP432 reference code for this program is: msp432p401x_p1_03

    d) [**Optional**] Read at least the first three sections of ARM Application Note 237 "Migrating from 8051 to Cortex Microcontrollers" which is available at http://infocenter.arm.com/help/index.jsp and on the course Canvas site.

    e) Demonstrate your programs and explain your key learnings to the TA's.

19. **[Part 2 Required Element]:** Demonstrate to the TA that you are using source code version control of your design files for this course (8051, ARM, etc.) and that those files are private. As indicated earlier in this lab assignment, "Students in this course need to be using a version control system for source code and other course design files. Options include Git, Subversion (SVN), or others. Students need to make their code for this course private and not share it with others."

20. **[Supplemental Element[1]]:** Design and implement a debug circuit for your 8051 board using a 74LS374 latch which allows values to be written to the 74LS374 chip whenever a write cycle is performed in "CODE / EPROM" address space (0000h–7FFFh). Note that for this lab the NVRAM will not be activated during a write cycle, since in this lab the NVRAM is simulating a non-writable EPROM. The latch must **not** activate for writes in the higher address range of 8000h-FFFFh. A debug latch like this can be used to help debug firmware by tracing function calls - each function could write a unique value to the 74LS374 latch via a standard Port 0 data write transfer, and the sequence of latch values could be seen using a logic analyzer.

    Devise a method for proving that your debug latch works correctly and that you can change its value under software control. One method is to write a value to the latch at the beginning of your ISR - that value should be incremented each time the ISR is executed, and will repetitively cycle through values from 80-FFh. A second value will be written to the latch inside of the main loop (non-ISR) - that value should be incremented each time the main loop is executed, and will repetitively cycle through values from 00-7Fh. Note that the two sections of code will always generate debug codes that are unique to their section of code (e.g. the ISR will never output codes in the range of 00-7Fh).

    The outputs of the 74LS374 should be constantly enabled (so that they're always driving valid data out) and these outputs can be left unconnected on your board. The outputs can be monitored using a logic analyzer. (They can be hooked up to LEDs, since the 74LS374 can drive more current than other ICs; however, this will increase the amount of power consumed by your board.) Some students may choose to hook up the '374 to a 7-segment LED display. Try to minimize power usage if you choose to use LEDs.

- **Perform a timing analysis to prove that your design satisfies the setup and hold requirements for the 74LS374. Your timing analysis should consist of two parts. First, calculate your circuit's minimum setup and hold time using the data sheets for the logic chips used in your design. Second, use a logic analyzer to measure the setup and hold time as seen at the '374 chip. Does your measured time satisfy the setup and hold time requirements of the '374? Discuss your timing analysis with the TA during signoff.**

[1] Required elements are necessary in order to meet the minimum requirements for the lab. Supplemental and challenge elements of the lab assignment may be completed by the student to qualify for a higher grade, but they do not have to be completed to successfully meet the minimum requirements for the lab.

## Submission Questions

21. **[Part 2 Required Element[1]]** As part of your submission, provide answers to the following:

a) What operating system (including revision) did you use for your 8051 code development?

b) What assembler(s) (including revision) did you use?

c) What ARM development tools did you use?

d) Did you install and use any other software tools to complete your lab assignment?

e) Did you experience any problems or challenges with this lab assignment or any of the software tools? If so, describe the issues.

f) If you have any suggestions for changes to this lab assignment for the future, please include those ideas in your submission.

## Submission Preparation

In addition to the items listed on the signoff checklist, be sure to review the lab for additional requirements for submission, including:

- ISR timing measurements and calculations shown on listing printout; $t_{LLPL}$ timing analysis**.**
- All code is well commented (both assembly and SPLD code), including header comments; printout (PDF) neat and easy to read.
- Scan of signed & dated signoff sheet with honor code pledge as the top sheet (No cover sheet please)
- (Scan of) answers to applicable lab questions (e.g. ISR timing calculations and measurements shown on .LST file, timing proof for $t_{LLPL}$, setup/hold time timing analysis, submission questions)
- Full copy of complete and accurate schematic of acceptable quality (all old/new components shown). Include programmable logic source code (e.g. .PLD file for the SPLD).
- Fully, neatly, and clearly commented code in .ASM and .LST files. Ensure your code is neat and easy to read, and that each source file has header comments that identify the author and any leveraged code the file contains. 8051 and ARM code should be in different folders and be well organized.

---

**NOTE:** Students are highly encouraged to start Lab #3 as soon as they finish Lab #2. Lab #3 is more complicated than Lab #2 and will take more time to complete. Students should also be making progress on their other assignments (e.g. final project) in parallel.

## Submission Instructions

Instructions: Print your name and sign the honor code pledge. Separate the signoff sheet from the rest of the lab and turn in a scan of the signed form, the items in the checklist below, and the answers to any applicable lab questions in order to receive credit for your work. No cover sheet please. **Submit all items electronically via Canvas (https://canvas.colorado.edu).**

Please follow the instructions given below:

1. Create a folder called "lastname_lab_2" where "lastname" is your last name. You will create sub-folders inside this unzipped folder, ending with a folder structure like the following:
    - lastname_lab_2
        - schematic
        - asm_code
        - spld_code
        - stm32_code
        - efm8
        - lab_writeup
2. Please include a legible and easy-to-view copy of your complete and accurate schematic (all old/new components shown) in pdf format inside the sub-folder named "schematic".
3. Include all your 8051 assembly code files like .asm and .lst in the sub-folder named "asm_code". Ensure that these files are neatly formatted and easy to read.
4. Include your spld source file (i.e. .pld file) and any spld simulation file (i.e. .si file) in the sub-folder named "spld_code".
5. Submit all of your ARM code files in the sub-folder named "stm32_code" (or alternate, depending on what development board you used). Ensure that these files are neatly formatted and easy to read.
6. Submit any EFM8BB1 code you demonstrated in the sub-folder named “efm8”.
7. Submit your write-up in the sub-folder named "lab_writeup" either in Word or PDF format. Your write-up should follow these guidelines:
    - Include scan of signed and dated signoff sheet as the first page. Include the backside of the signoff sheet as well if there are comments written by the TAs. Do not add any cover sheet.
    - Write-up should be concise and less than 5 pages (excluding the signoff sheet). Use only font size 11 or 12 to maintain legibility. Do not describe the objectives of the lab or any redundant information.
    - Include any important analyses and timing diagrams
    - Include all the calculations that are required to be done for answering the questions that have been asked in the sign-off sheets or lab assignment.
    - Include the screen-shots (if any) in PDF format in this folder itself.
    - Comment on any significant learnings from the lab assignment.
    - Include the answers to any submission questions in the lab assignment.
    - Must include clear high-resolution pictures of top and bottom sides of the 8051 board you assembled for this lab, including wiring and labels; wire wrapping and soldering should be clearly visible – **make sure your pictures are in focus**. Use JPEG format.
8. Submit a zipped version of the folder as "lastname_lab_2.zip" as an attachment via the Canvas interface. Please use only the “.zip” file format when submitting files.
9. Please contact the TA’s or instructor in case you have any doubts regarding submissions.

**NOTE: Make and save archive copies of your assembly/SPLD code and schematic files. You need to submit the Lab #2 files electronically, now and at the end of the semester.**

**ECEN 5613** **Embedded System Design** **Fall 2023**

**Lab #2 Signoff Sheet**

You will need to obtain the signature of your instructor or TA on the following items in order to receive credit for your lab assignment. Signatures are due by **Friday, September 29, 2023 (Part 1 Elements) and Friday, October 6, 2023 (Part 2 Elements)**.

Print your name below, sign the honor code pledge, and then demonstrate your working hardware & firmware in order to obtain the necessary signatures.

**Student Name: _____**

**Honor Code Pledge**: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

**Student Signature: _____**

**Signoff Checklist**

Part 1 Required Elements

- ❑ Schematic of acceptable quality, correct memory map, SPLD .PLD file
- ❑ Pins and signals labeled, decoupling capacitors, and two 28-pin wire wrap sockets present on board
- ❑ NVRAM (as EPROM substitute), decode logic, and LED functional
- ❑ Understands device programmer.
- ❑ Demonstrated ability to use logic analyzer to capture bus cycles and view fetches from NVRAM. Shows detailed knowledge of both state and timing modes. Captures latched address lines A[15:0], data lines D[7:0], ALE, /PSEN, and NVRAM chip select signal on the logic analyzer display.
- ❑ Shows and discusses logic analyzer screen captures:
- ❑ Assembly program and timer ISR functional:

_____

**TA signature and date**

Part 2 Required and Supplemental Elements

- ❑ AT89C51RC2, RS-232, and FLIP functional
- ❑ 74LS374 debug port functional
- ❑ Understands timing analysis, setup/hold/propagation
- ❑ ARM code build process, LED program, version control

_____

**Instructor/TA Comments:** □ □ □ **TA signature and date**

| **FOR INSTRUCTOR USE ONLY**<br>**Part 1 Elements** | Not Applicable | Poor/Not Complete | Meets Requirements | Exceeds Requirements | Outstanding |
|---|---|---|---|---|---|
| Schematics, SPLD code | ☐ | ☐ | ☐ | ☐ | ☐ |
| Hardware physical implementation | ☐ | ☐ | ☐ | ☐ | ☐ |
| Part 1 Required Elements functionality | ☐ | ☐ | ☐ | ☐ | ☐ |
| Sign-off done without excessive retries | | ☐ | ☐ | | |
| Student understanding and skills | ☐ | ☐ | ☐ | ☐ | ☐ |
| Overall Demo Quality (Part 1 Elements) | ☐ | ☐ | ☐ | ☐ | ☐ |

| **FOR INSTRUCTOR USE ONLY**<br>**Part 2 Elements** | Not Applicable | Poor/Not Complete | Meets Requirements | Exceeds Requirements | Outstanding |
|---|---|---|---|---|---|
| Schematics, SPLD code | ☐ | ☐ | ☐ | ☐ | ☐ |
| Hardware physical implementation | ☐ | ☐ | ☐ | ☐ | ☐ |
| Part 2 Required Elements functionality | ☐ | ☐ | ☐ | ☐ | ☐ |
| Supplemental Elements functionality | ☐ | ☐ | ☐ | ☐ | ☐ |
| Sign-off done without excessive retries | | ☐ | ☐ | | |
| Student understanding and skills | ☐ | ☐ | ☐ | ☐ | ☐ |
| Overall Demo Quality (Part 2 Elements) | ☐ | ☐ | ☐ | ☐ | ☐ |

**NOTE: This signoff sheet should be the top/first sheet of your submission.**